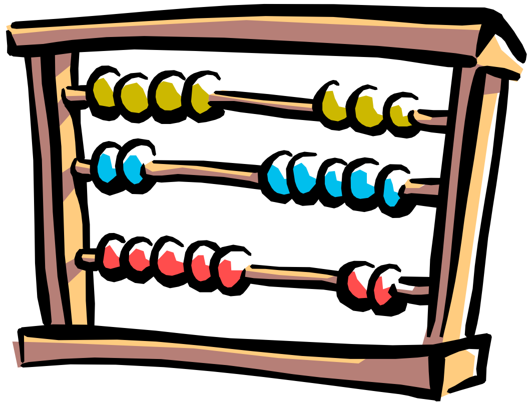
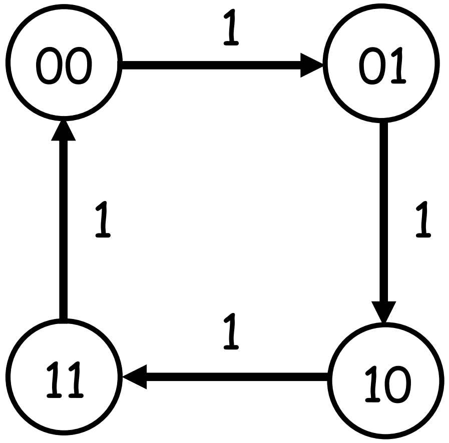
# Counters

# Counters

- Counters are a specific type of sequential circuit.
- Like registers, the state, or the flip-flop values themselves, serves as the "output."
- The output value increases by one on each clock cycle.
- After the largest value, the output "wraps around" back to 0.
- Using two bits, we'd get something like this:



| Present State | | Next State | |
|---|---|---|---|
| A | B | A | B |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |



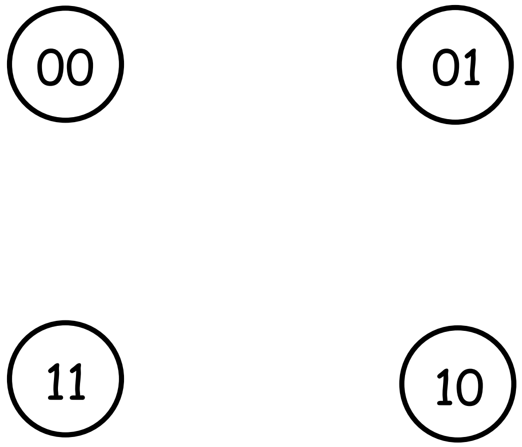
$$\begin{tikzcd}
00 \arrow[r, "1"] & 01 \arrow[d, "1"] \\
11 \arrow[u, "1"] & 10 \arrow[l, "1"]
\end{tikzcd}$$

## Benefits of counters

- Counters can act as simple clocks to keep track of “time.”
- You may need to record how many times something has happened.
  - How many bits have been sent or received?
  - How many steps have been performed in some computation?
- All processors contain a program counter, or PC.
  - Programs consist of a list of instructions that are to be executed one after another (for the most part).
  - The PC keeps track of the instruction currently being executed.
  - The PC increments once on each clock cycle, and the next program instruction is then executed.

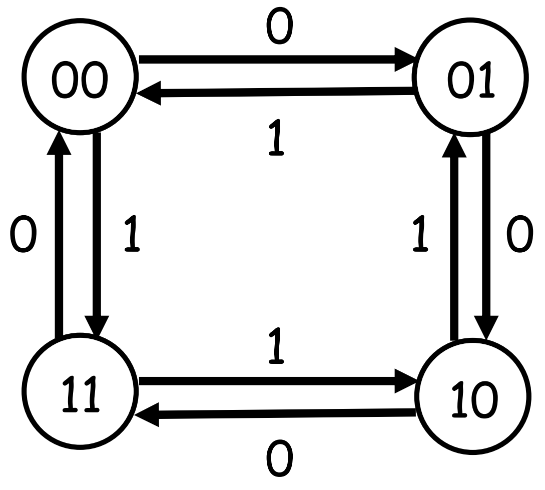
## A slightly fancier counter

- Let's try to design a slightly different two-bit counter:
  - Again, the counter outputs will be 00, 01, 10 and 11.
  - Now, there is a single input, X. When X=0, the counter value should *increment* on each clock cycle. But when X=1, the value should *decrement* on successive cycles.
- We'll need two flip-flops again. Here are the four possible states:



## The complete state diagram and table

- Here's the complete state diagram and state table for this circuit.



| Present State | | Inputs | Next State | |
|---|---|---|---|---|
| $Q_1$ | $Q_0$ | X | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

## D flip-flop inputs

- If we use D flip-flops, then the D inputs will just be the same as the desired next states.
- Equations for the D flip-flop inputs are shown at the right.
- Why does $D_0 = Q_0'$ make sense?

| Present State | | Inputs | Next State | |
|---|---|---|---|---|
| $Q_1$ | $Q_0$ | X | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

| | | | | |
|---|---|---|---|---|
| | | | $Q_0$ | $Q_0$ |
| | 0 | 1 | 0 | 1 |
| $Q_1$ | 1 | 0 | 1 | 0 |
| | | X | X | |

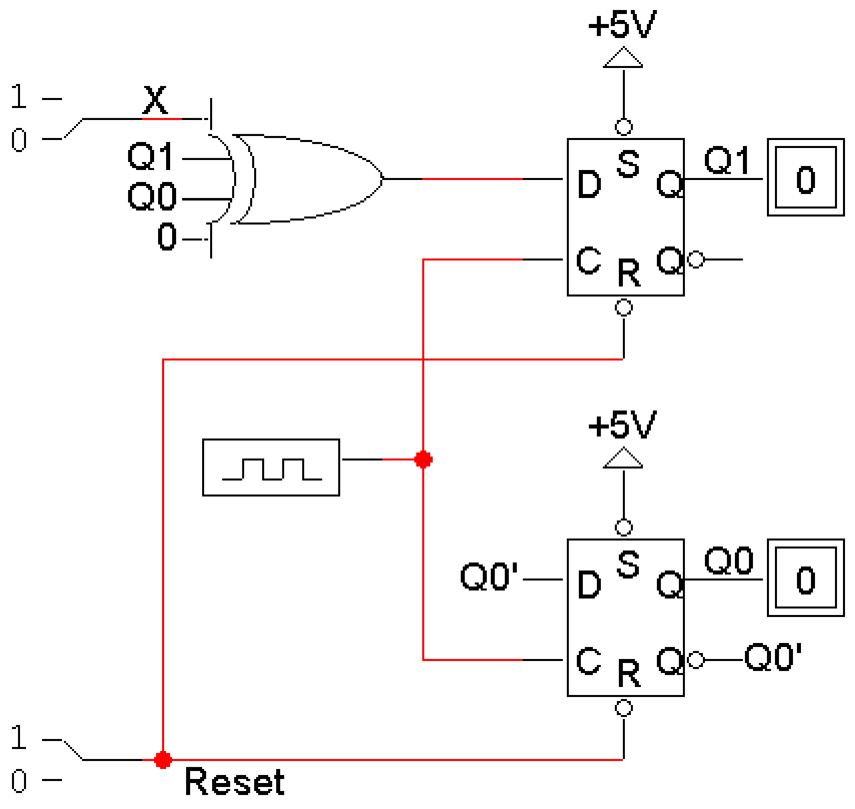
$D_1 = Q_1 \oplus Q_0 \oplus X$

| | | | | |
|---|---|---|---|---|
| | | | $Q_0$ | $Q_0$ |
| | 1 | 1 | 0 | 0 |
| $Q_1$ | 1 | 1 | 0 | 0 |
| | | X | X | |

$D_0 = Q_0'$

# The counter in LogicWorks

- Here are some D Flip Flop devices from LogicWorks.
- They have both normal and complemented outputs, so we can access Q0' directly without using an inverter. (Q1' is not needed in this example.)
- This circuit counts normally when Reset = 1. But when Reset is 0, the flip-flop outputs are cleared to 00 immediately.
- There is no three-input XOR gate in LogicWorks so we've used a four-input version instead, with one of the inputs connected to 0.



# JK flip-flop inputs

- If we use JK flip-flops instead, then we have to compute the JK inputs for each flip-flop.
- Look at the present and desired next state, and use the excitation table on the right.

| Q(t) | Q(t+1) | J | K |
|---|---|---|---|
| 0 | 0 | 0 | x |
| 0 | 1 | 1 | x |
| 1 | 0 | x | 1 |
| 1 | 1 | x | 0 |

| Present State | | Inputs | Next State | | Flip flop inputs | | | |
|---|---|---|---|---|---|---|---|---|
| $Q_1$ | $Q_0$ | X | $Q_1$ | $Q_0$ | $J_1$ | $K_1$ | $J_0$ | $K_0$ |
| 0 | 0 | 0 | 0 | 1 | 0 | x | 1 | x |
| 0 | 0 | 1 | 1 | 1 | 1 | x | 1 | x |
| 0 | 1 | 0 | 1 | 0 | 1 | x | x | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | x | x | 1 |
| 1 | 0 | 0 | 1 | 1 | x | 0 | 1 | x |
| 1 | 0 | 1 | 0 | 1 | x | 1 | 1 | x |
| 1 | 1 | 0 | 0 | 0 | x | 1 | x | 1 |
| 1 | 1 | 1 | 1 | 0 | x | 0 | x | 1 |

## JK flip-flop input equations

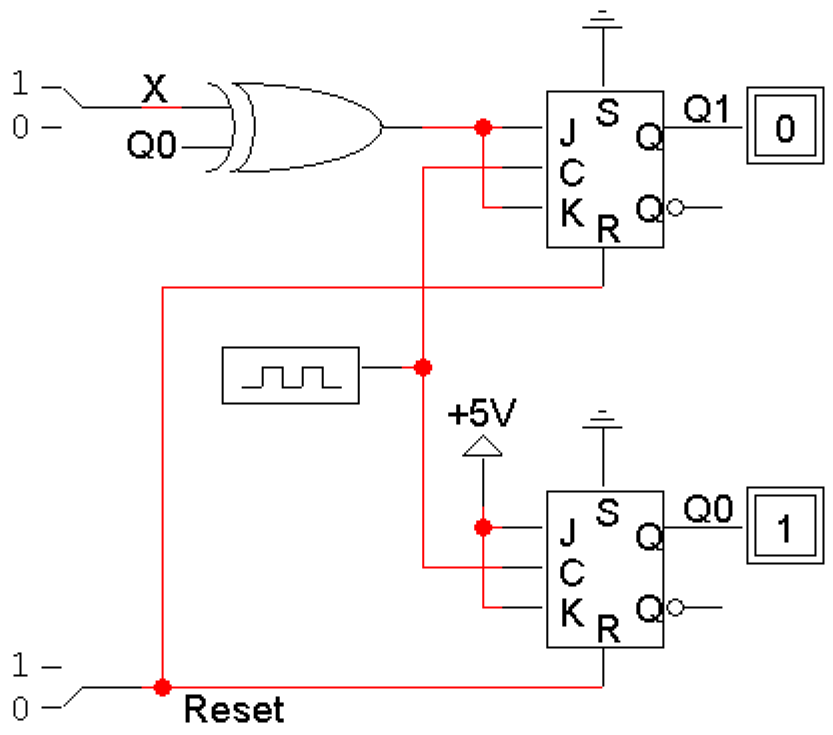
| Present State | | Inputs | Next State | | Flip flop inputs | | | |
|---|---|---|---|---|---|---|---|---|
| $Q_1$ | $Q_0$ | X | $Q_1$ | $Q_0$ | $J_1$ | $K_1$ | $J_0$ | $K_0$ |
| 0 | 0 | 0 | 0 | 1 | 0 | x | 1 | x |
| 0 | 0 | 1 | 1 | 1 | 1 | x | 1 | x |
| 0 | 1 | 0 | 1 | 0 | 1 | x | x | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | x | x | 1 |
| 1 | 0 | 0 | 1 | 1 | x | 0 | 1 | x |
| 1 | 0 | 1 | 0 | 1 | x | 1 | 1 | x |
| 1 | 1 | 0 | 0 | 0 | x | 1 | x | 1 |
| 1 | 1 | 1 | 1 | 0 | x | 0 | x | 1 |

- We can then find equations for all four flip-flop inputs, in terms of the present state and inputs. Here, it turns out $J_1 = K_1$ and $J_0 = K_0$.

$$J_1 = K_1 = Q_0' X + Q_0 X'$$
$$J_0 = K_0 = 1$$

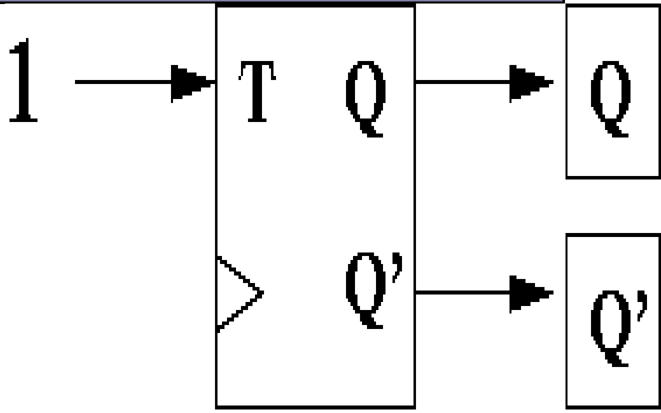
# The counter in LogicWorks again

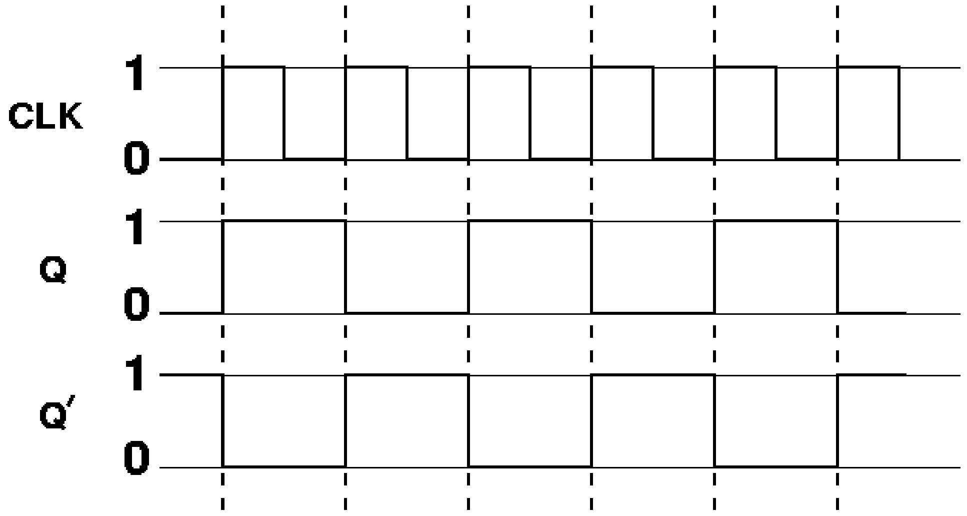
- Here is the counter again, but using JK Flip Flop .
- The direct inputs R and S are non-inverted, or active-high.
- So this version of the circuit counts normally when Reset = 0, but initializes to 00 when Reset is 1.



## Asynchronous Counters

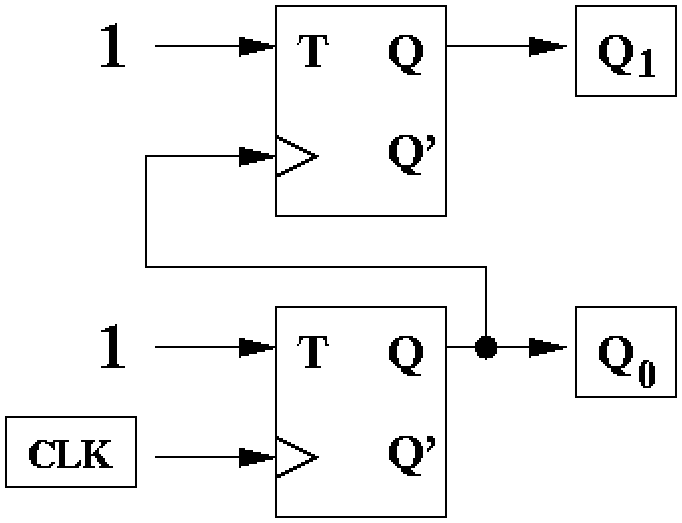
- This counter is called *asynchronous* because not all flip flops are hooked to the same clock.
- Look at the waveform of the output, **Q**, in the timing diagram. It resembles a clock as well. If the period of the clock is T, then what is the period of **Q**, the output of the flip flop? It's 2T!
- We have a way to create a clock that runs twice as slow. We feed the clock into a T flip flop, where T is hardwired to 1. The output will be a clock who's period is twice as long.





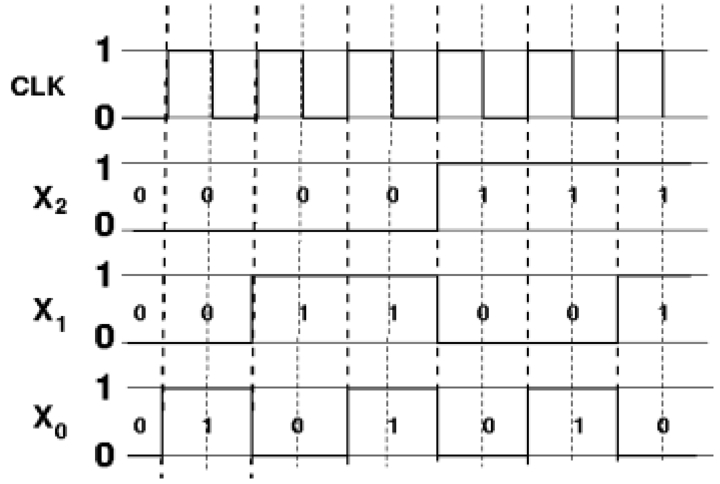
## Asynchronous counters

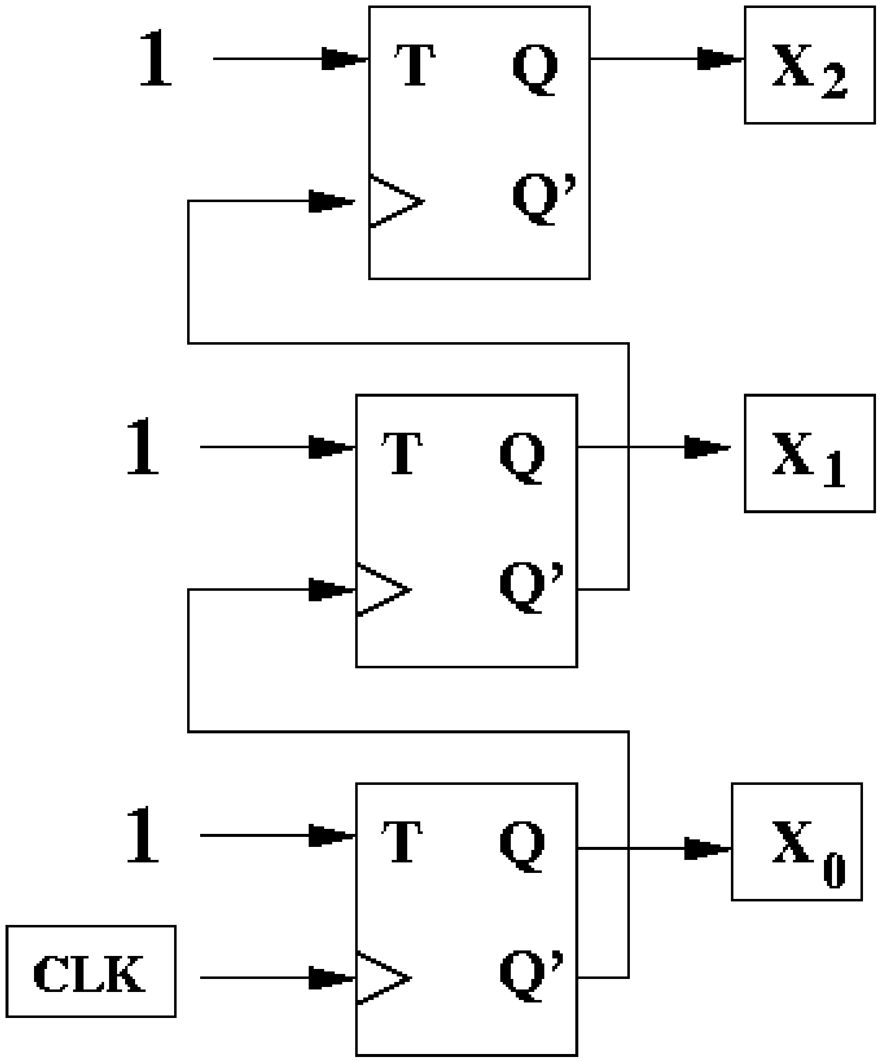
If the clock has period T. **Q0** has period 2T. **Q1** period is 4T
With n flip flops the period is $2^n$.



## 3 bit asynchronous “ripple” counter using T flip flops

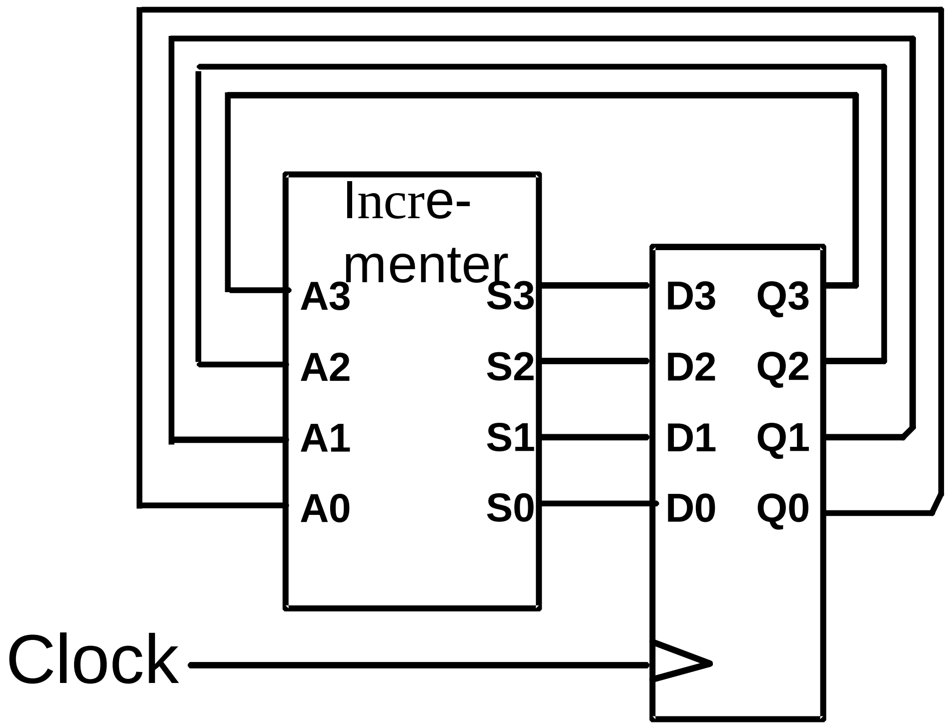
- This is called as a *ripple counter* due to the way the FFs respond one after another in a kind of rippling effect.





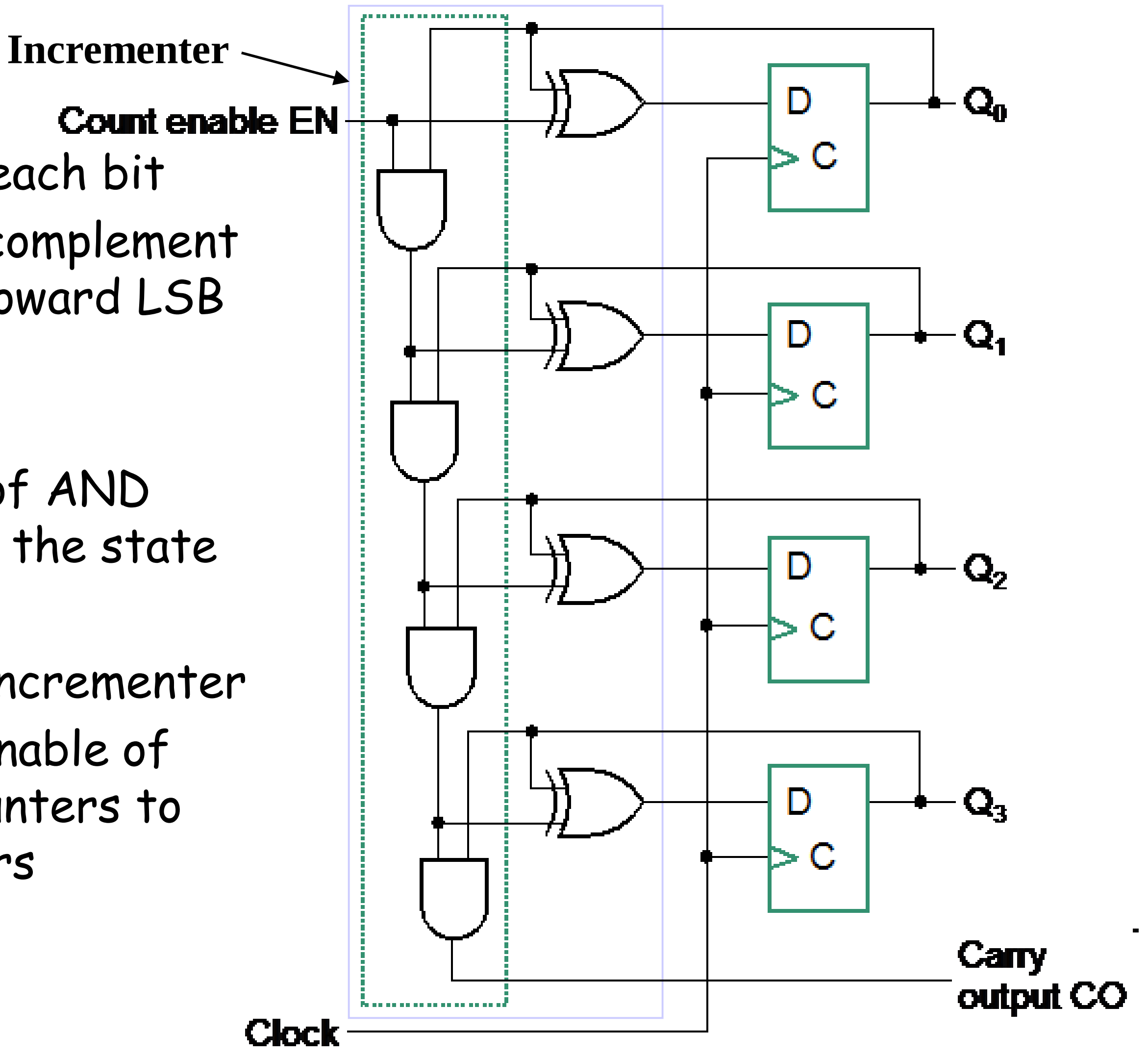
## Synchronous Counters

- To eliminate the "ripple" effects, use a common clock for each flip-flop and a combinational circuit to generate the next state.
- For an up-counter, use an incrementer =>



# Synchronous Counters (continued)

- Internal details =>
- Internal Logic
  - XOR complements each bit
  - AND chain causes complement of a bit if all bits toward LSB from it equal 1
- Count Enable
  - Forces all outputs of AND chain to 0 to "hold" the state
- Carry Out
  - Added as part of incrementer
  - Connect to Count Enable of additional 4-bit counters to form larger counters



(a) Logic Diagram-Serial Gating

## Design Example: Synchronous BCD

- Use the sequential logic model to design a synchronous BCD counter with D flip-flops
- State Table =>
- Input combinations 1010 through 1111 are don't cares

| Current State | | | | Next State | | | |
|---|---|---|---|---|---|---|---|
| Q8 | Q4 | Q2 | Q1 | Q8 | Q4 | Q2 | Q1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

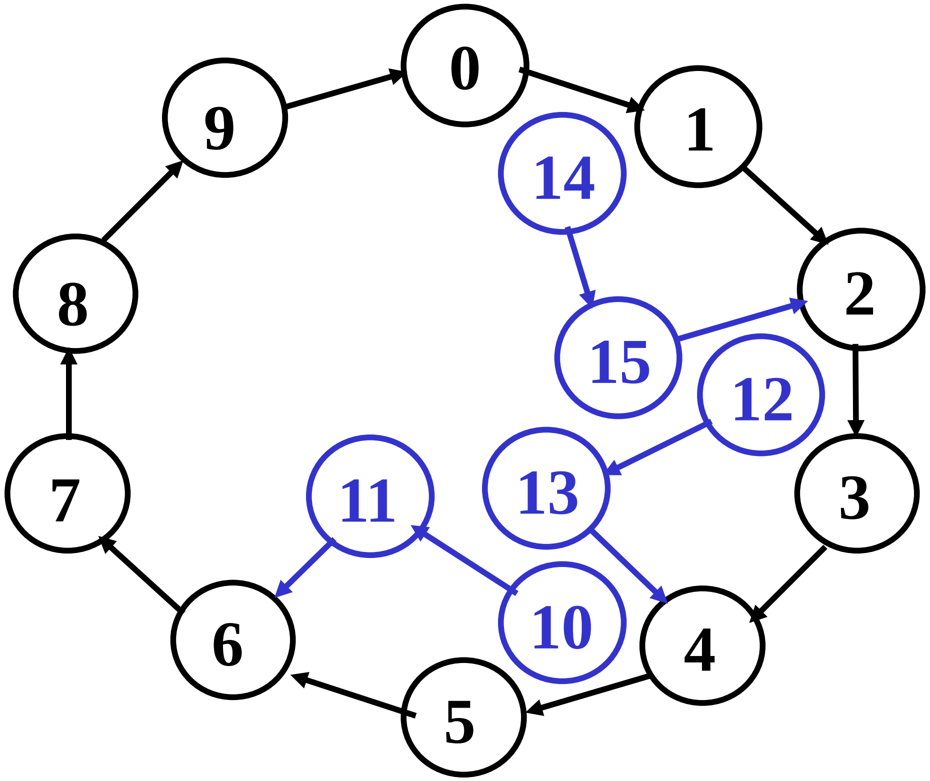
## Synchronous BCD (continued)

- Use K-Maps to two-level optimize the next state equations and manipulate into forms containing XOR gates:
  $D1 = Q1'$
- $D2 = Q2 \oplus Q1Q8'$
  $D4 = Q4 \oplus Q1Q2$
  $D8 = Q8 \oplus (Q1Q8 + Q1Q2Q4)$
  - $Y = Q1Q8$
- The logic diagram can be drawn from these equations
  - An asynchronous or synchronous reset should be added
- What happens if the counter is perturbed by a power disturbance or other interference and it enters a state other than 0000 through 1001?

## Synchronous BCD (continued)

- Find the actual values of the six next states for the don't care combinations from the equations
- Find the overall state diagram to assess behavior for the don't care states (states in decimal)

| Present State | Next State |
|---|---|
| Q8 Q4 Q2 Q1 | Q8 Q4 Q2 Q1 |
| 1 0 1 0 | 1 0 1 1 |
| 1 0 1 1 | 0 1 1 0 |
| 1 1 0 0 | 1 1 0 1 |
| 1 1 0 1 | 0 1 0 0 |
| 1 1 1 0 | 1 1 1 1 |
| 1 1 1 1 | 0 0 1 0 |



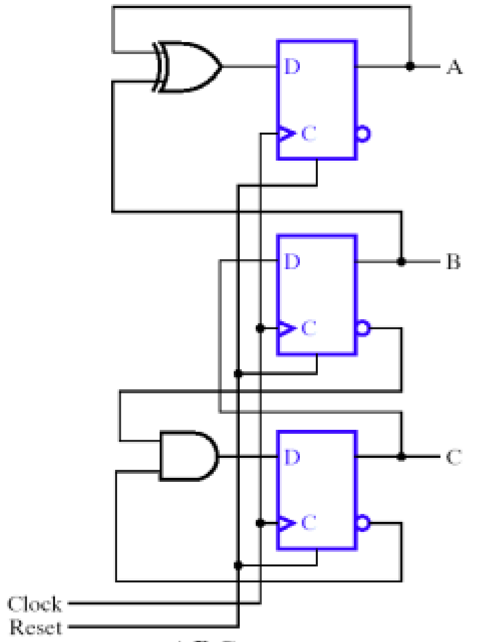
## **Synchronous BCD** (continued)

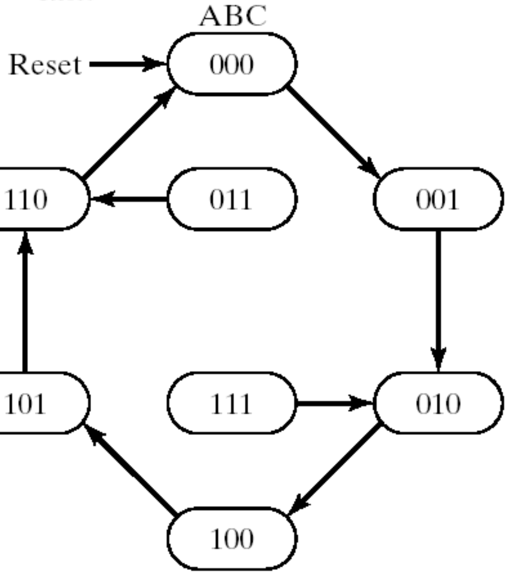
- For the BCD counter design, if an invalid state is entered, return to a valid state occurs within two clock cycles
- Is this adequate?!

# Counting an arbitrary sequence

- **TABLE 7-10**
  **State Table and Flip-Flop Inputs for Counter**

| Present State | | | Next State | | |
|---|---|---|---|---|---|
| A | B | C | DA = A(t+1) | DB = B(t+1) | DC= C(t+1) |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |

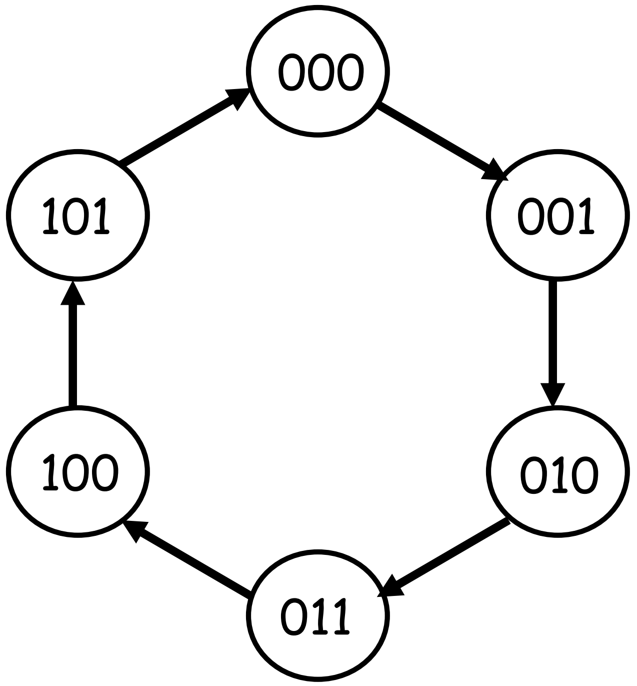




## Unused states

- The examples shown so far have all had $2^n$ states, and used n flip-flops. But sometimes you may have unused, leftover states.
- For example, here is a state table and diagram for a counter that repeatedly counts from 0 (000) to 5 (101).
- What should we put in the table for the two unused states?

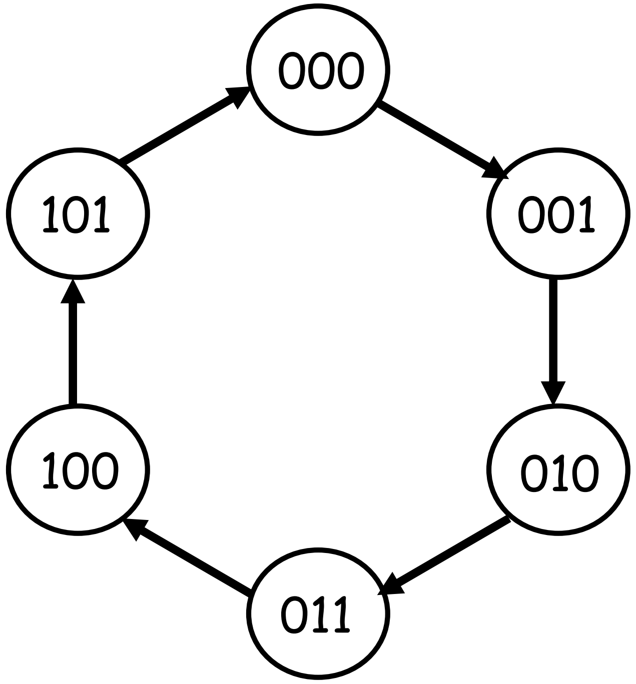
| Present State | | | Next State | | |
|---|---|---|---|---|---|
| $Q_2$ | $Q_1$ | $Q_0$ | $Q_2$ | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | ? | ? | ? |
| 1 | 1 | 1 | ? | ? | ? |



## Unused states can be don't cares...

- To get the *simplest* possible circuit, you can fill in don't cares for the next states. This will also result in don't cares for the flip-flop inputs, which can simplify the hardware.
- If the circuit somehow ends up in one of the unused states (110 or 111), its behavior will depend on exactly what the don't cares were filled in with.

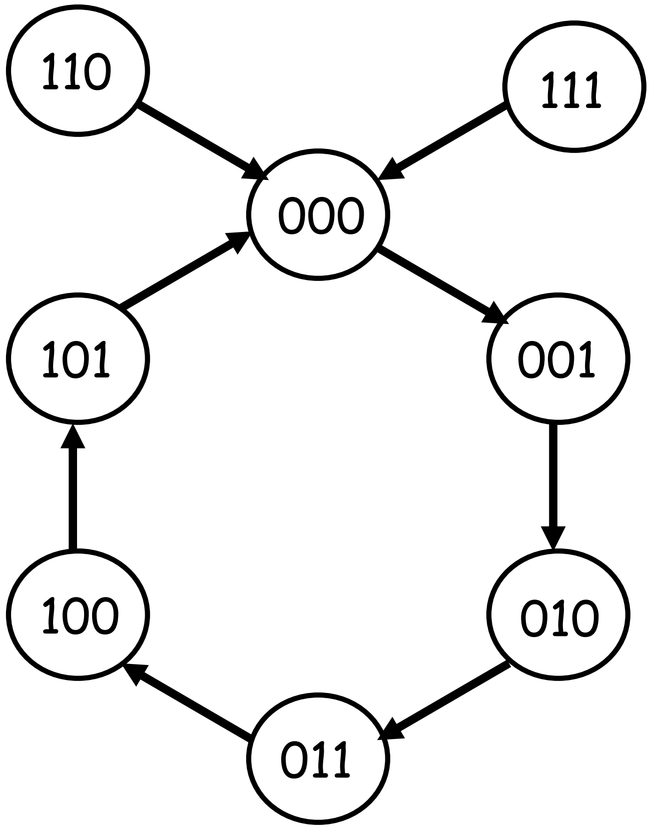
| Present State | | | Next State | | |
|---|---|---|---|---|---|
| $Q_2$ | $Q_1$ | $Q_0$ | $Q_2$ | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | x | x | x |
| 1 | 1 | 1 | x | x | x |



## …or maybe you *do* care

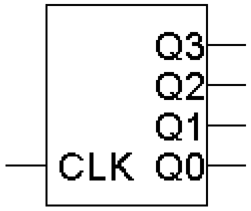
- To get the *safest* possible circuit, you can explicitly fill in next states for the unused states 110 and 111.
- This guarantees that even if the circuit somehow enters an unused state, it will eventually end up in a valid state.
- This is called a self-starting counter.

| Present State | | | Next State | | |
|---|---|---|---|---|---|
| $Q_2$ | $Q_1$ | $Q_0$ | $Q_2$ | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |



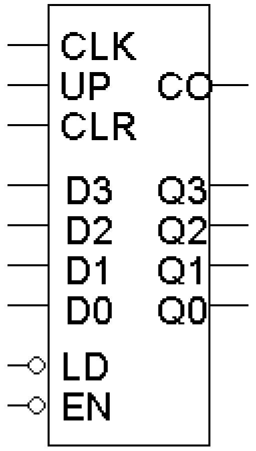
## LogicWorks counters

- There are a couple of different counters available in LogicWorks.
- The simplest one, the Counter-4 Min, just increments once on each clock cycle.
  - This is a four-bit counter, with values ranging from 0000 to 1111.
  - The only "input" is the clock signal.



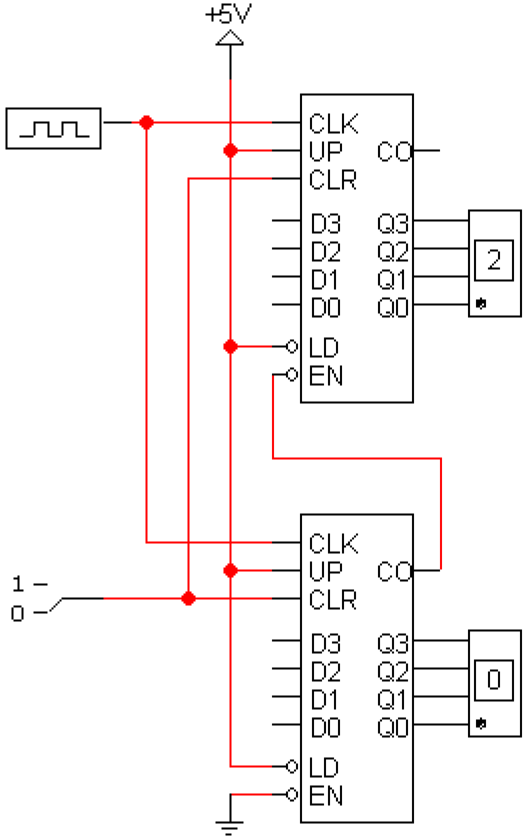
# More complex counters

- More complex counters are also possible. The full-featured LogicWorks Counter-4 device below has several functions.
  - It can increment or decrement, by setting the UP input to 1 or 0.
  - You can immediately (asynchronously) clear the counter to 0000 by setting CLR = 1.
  - You can specify the counter's next output by setting $D_3$-$D_0$ to any four-bit value and clearing LD.
  - The active-low EN input enables or disables the counter.
    - When the counter is disabled, it continues to output the same value without incrementing, decrementing, loading, or clearing.
  - The "counter out" CO is normally 1, but becomes 0 when the counter reaches its maximum value, 1111.



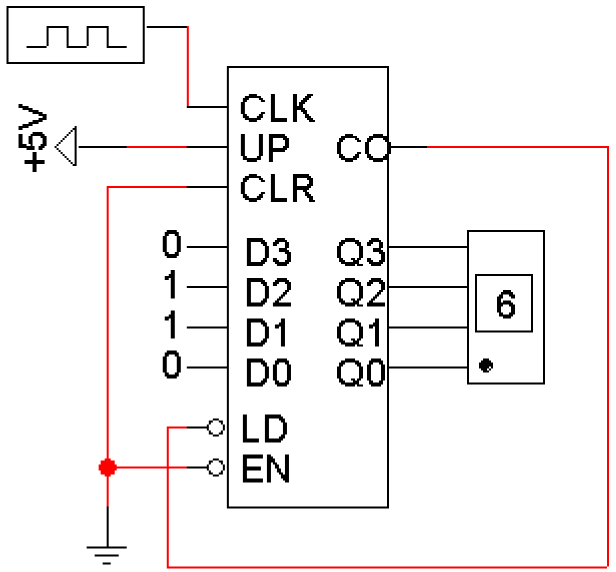
# An 8-bit counter

- As you might expect by now, we can use these general counters to build other counters.
- Here is an 8-bit counter made from two 4-bit counters.
  - The bottom device represents the least significant four bits, while the top counter represents the most significant four bits.
  - When the bottom counter reaches 1111 (i.e., when CO = 0), it enables the top counter for one cycle.
- Other implementation notes:
  - The counters share clock and clear signals.



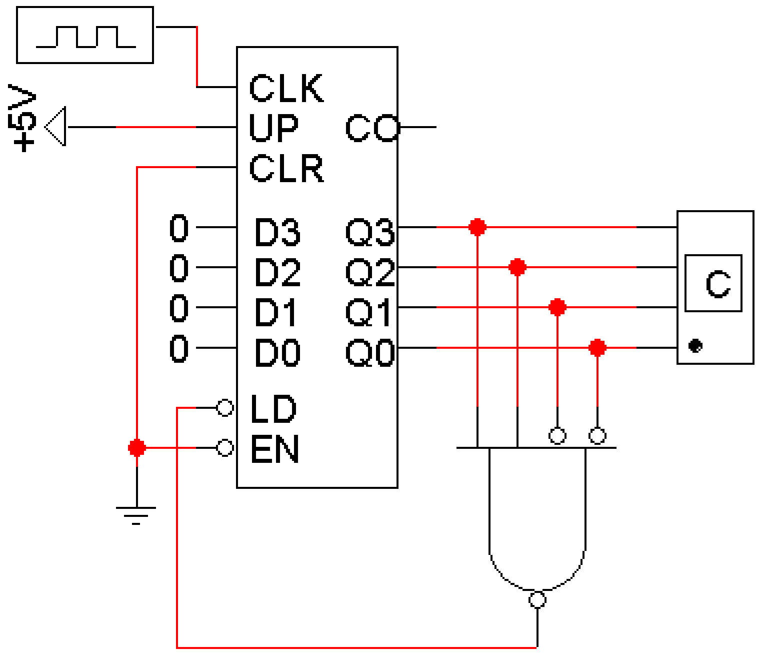
# A restricted 4-bit counter

- We can also make a counter that “starts” at some value besides 0000.
- In the diagram below, when CO=0 the LD signal forces the next state to be loaded from $D_3$-$D_0$.
- The result is this counter wraps from 1111 to 0110 (instead of 0000).

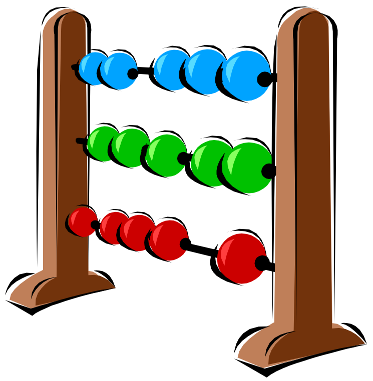


## Another restricted counter

- We can also make a circuit that counts up to only 1100, instead of 1111.
- Here, when the counter value reaches 1100, the NAND gate forces the counter to load, so the next state becomes 0000.



## Summary of Counters



- Counters serve many purposes in sequential logic design.
- There are lots of variations on the basic counter.
  - Some can increment or decrement.
  - An enable signal can be added.
  - The counter's value may be explicitly set.
- There are also several ways to make counters.
  - You can follow the sequential design principles to build counters from scratch.
  - You could also modify or combine existing counter devices.